# XML *Digital Signature* pada *wireless Web services*

Febrian Setiadi – NIM : 13503028

*Program Studi Teknik Informatika, Institut Teknologi Bandung*
*Jl. Ganesha 10, Bandung*
E-mail : febrian.setiadi@gmail.com

## Abstraksi

Saat ini *Web services* menjadi sangat populer di *enterprise* karena kemampuannya dalam mengintegrasikan aplikasi-aplikasi yang berbeda *plattform* dan mampu memperbaiki kelemahan dari *middleware* konvensional. Saat sebuah *enterprise* ingin mengintegrasikan sistem bisnis dengan partnernya menggunakan internet, maka informasi yang dialirkan harus dipastikan dalam keadaan aman. Oleh karena itu keamanan menjadi isu yang sangat penting dalam implementasi *Web services*.

Dalam Makalah ini akan dibahas aspek keamanan dalam *wireless Web services. Wireless web services* terdiri dari dua *plattform* berbeda, salah satunya adalah *wireless device.* Lalu dijelaskan penggunaan dari *XML digital signature*. Secara umum penjelasannya mencakup penanganan *digital-signature XML* dalam aplikasi di sisi wireless, dan bagaimana *digital-signature XML* diproses di sisi *Web services*.

Dalam implementasinya digunakan *API Bouncy Castle Java cryptography package* untuk menangani *digital signatures*. *Bouncy Castle* dipilih karena kinerjanya yang cukup bagus dalam lingkungan pengembangan *wireless device* (*MIDP*), dan sudah menyediakan banyak modul-modul pendukung dalam kriptografi.

Teknologi *XML digital signature* dapat diterapkan dalam sebuah perangkat keamanan dalam aplikasi *wireless Web services*. Dalam Makalah ini akan diberikan contoh penggunaan dari pengamanan sebuah *XML message* yang dipertukarkan antara sisi *front-end*, *wireless* (*J2ME/MIDP*) dan sisi *back-end* yaitu halaman *JSP*.

**Kata kunci**: *wireless Web services, XML digital signature, API Bouncy-Castle.*

## A. Pendahuluan

Saat ini *mobile application* telah banyak digunakan dan telah banyak *wireless Web services* yang sudah menggunakan *plattform* mobile sebagai salah satu *plattform* utamanya. Apalagi dalam beberapa dekade kedepan, komunikasi data lewat *mobile device* dan sebaliknya akan jauh lebih mudah seiring diluncurkannya teknologi 3G yang memnungkinkan pengiriman data dengan kecepatan tinggi.

Anda mungkin sudah sering mengirim dokumen bisnis yang di-*attach* pada *e-mail* melalui Internet. Efisien, cepat dan murah. Tapi mungkin kita lupa bahwa Internet adalah suatu *public network* yang tidak aman. Saat pengiriman dokumen, seseorang bisa saja dengan ilegal mengubah isi dokumen itu tanpa diketahui pengirim atau penerima. Tanpa fasilitas keamanan yang baik, sang penerima akan menerima dokumen tersebut tanpa mencurigai

adanya perubahan. Namun jika pengirim membubuhkan tanda tangan digital pada dokumen itu, penerima dapat merasa yakin bahwa setelah ditandatangani pengirim, dokumen itu tidak ada yang memanipulasi saat dalam perjalanan, karena ada proses otentikasi yang benar-benar memastikan keotentikan pesan.

Dalam *wireless Web services* jika seseorang ingin mengakses suatu layanan dari *web services*, harus dipastikan bahwa setiap konten yang ingin didapatkan adalah benar otentik dan masih orisinal, hal ini bisa diatasi dengan penggunaan tanda tangan digital pada setiap konten yang ingin diakses melalui *mobile-device*.

Teknologi *XML digital signature* bisa digunakan sebagai solusi keamanan dari aplikasi *wireless web services*. Dalam makalah ini akan dijelaskan penggunaan dari *XML digital signature* dan contoh aplikasi sederhananya. Aplikasi yang akan dibuat secara umum menjelaskan bagaimana pertukaran data *XML* antara dua *plattform* berbeda yang aman.

## 1. Teknologi Java dalam *wireless Web services*

*Java-based Web services* dan *wireless Java developmen*t adalah isu utama dalam pengembangan *wireless web Services.* Dalam prakteknya menggunakan *J2ME* pada sisi klien dan halaman *JSP* pada sisi *server*. *Web services* tidak terlalu dependen dan cenderung modular, pertukaran datanya menggunakan protokol komunikasi *XML* yang standar. Penggunaan dari *Web services* memungkinkan sebuah *vendor* untuk menawarkan berbagai layanannya dalam sebuah market pasar tertentu, sehingga pelanggan bisa memilih untuk membeli layanan dari *vendor* yang berbeda, tergantung kebutuhannya. Situasi seperti ini bisa dikatakan bahwa *Web services* cocok untuk melayani pelanggan yang dalam hal ini adalah *wireless front-end*.

*Plattform* Java memegang peranan penting dalam pembangunan aplikasi *wireless Web services*. Di sisi klien (*front-end*) digunakan teknologi *J2ME* yang menawarkan kompatibilitas terhadap hampir semua *mobile-devices*, dan *library* yang cukup banyak yang dapat digunakan untuk mngembangkan sebuah aplikasi *wireless Web services* di sisi klien yang cukup kompleks. Komponen utama dari *J2ME* adalah *MIDP* (*Mobile Information Device Profile*), yang menentukan *API* dari Java dan lingkungan pengembangan dalam sebuah *mobile device.* Karena banyak *low-end devices* yang kompatibel dengan *J2ME*, maka dalam pengembangan kedepannya teknologi ini akan semakin luas cakupannya. Di *plattform* lainnya, sisi *Web services*, digunakan teknologi *J2EE*, secara umum *J2EE* dikembangkan untuk *Web services*, karena *J2EE* menyediakan *API* dan sejumlah *library* untuk menangani pesan *XML* dari *Web services*. Fungsi utama dari *J2EE* diimplementasikan dalam teknologi *EJB* (*Enterprise Java Beans*), *API JDBC* (untuk konektifitas *database*), dan *API RMI* (*Remote Method Invocation*), kesemua teknologi yang ada dalam java tersebut dapat digunakan untuk membuat sebuah *Web services gateway*. Disamping itu digunakan juga *JSR172 (J2ME Web Services specification*), didalam spesifikasi tersebut dijelaskan kelas-kelas utama yang digunakan dalam sisi *wireless front-end* untuk pengembangan *wireless Web services.*

## 2. Isu keamanan dalam *wireless Web services*

Meski *Java-based wireless Web services* akan berkembang pesat dalam dunia *mobile-commerce*, teknologi ini belum bisa dikatakan matang. Keamanan adalah salah satu isu utama untuk diperhatikan. Komunikasi *wireless* adalah target utama dari penyadapan dan intersepsi gelombang udara, dan kebanyakan *wireless device* belum bisa melakukan sebuah mekanisme pengamanan berupa penyandian keseluruhan dari data yang ditransmisikan selama komunikasi. Lagipula di sisi *back-end*, *Web-services* biasanya berjalan diluar *firewall* perusahaan, dan berkomunikasi dengan protokol pengiriman pesan yang terbuka. Maka itu, *Wireless web services* adalah mudah diserang oleh berbagai macam jenis penyerangan. Salah satu protokol komunikasi *web-based* yang sifatnya *point-to-point* seperti *SSL/TLS* dan *HTTPS* tidak cocok diterapkan untuk aplikasi yang melibatkan banyak *vendor*, dan banyak perantara dalam jaringan *web services itu sendiri*. Fokus utamanya adalah pengamanan dari konten itu sendiri, bukan kepada jalur komunikasi yang dilalui untuk mencapai tujuan.

Untuk mengamankan konten yang ditransmisikan, bisa digunakan *XML digital signature*, dengan ini bisa dijamin bahwa data tetap otentik, dan memenuhi salah satu isu keamanan *web-services*. Di bagian selanjutnya dalam makalah ini akan ditunjukkan bagaimana mengimplementasikan *XML digital signature*

menggunakan plattform *J2ME/MIDP* dalam sisi *wireless-end* dan teknologi *JavaServer Pages* (JSP) di sisi *back-end*.

**3. Sifat-sifat dari komunikasi yang aman**

Integritas data hanya salah satu aspek dari pengamanan komunikasi. Tanda-tangan digital bisa menyelesaikan isu-isu lain juga dalam komunikasi digital. Secara umum jaringan komunikasi yang aman harus memenuhi kriteria sebagai berikut:

1. Keabsahan Pengirim

Hal ini berkaitan dengan kebenaran identitas pengirim. Dengan kata lain, masalah ini dapat diungkapkan sebagai pertanyaan: "Apakah pesan yang diterima benar-benar berasal dari pengirim yang sesungguhnya?". Sekelompok yang berhak melakukan komunikasi harus mengenali setiap anggotanya.

2. Keaslian pesan

Harus dapat dipastikan bahwa konten yang ditransmisikan tidak berubah. Tanda-tangan digital bisanya digunakan untuk memastikan integritas data.

3. Kerahasiaan pesan

Kadang, data komunikasi harus dirahasiakan juga. Tanda-tangan digital tidak memenuhi aspek kerahasiaan data. Untuk menjamin kerahasiaan data digunakan penyandian data.

4. Anti-penyangkalan

Ketika pesan dikirim, pengirim tidak dapat menyangkal bahwa dia tidak pernah mengirim pesan tersebut. Jika pesan yang dikirim telah ditanda-tangani, pengirim tidak dapat menyangkalnya, karena hanya dia yang bisa membuat tanda-tangan itu, hal ini biasanya terjadi karena kunci privat hanya dimiliki satu orang untuk membuat tanda-tangan digital.

**4. Mengamankan data dengan tanda-tangan digital**

Misalkan suatu saat seorang pialang saham menggunakan *mobile-device* nya untuk mengamati perubahan harga saham ketika dia sedang tidak berada di lantai bursa. Di tengah perjalanan dia mengakses layanan *web services* untuk mengamati harga saham yang sedang naik. Dan dari konten yang diterimanya terlihat bahwa harga saham tertentu yang sedang dalam perhatiannya turun drastis dibawah nilai seharusnya. Haruskah dia membeli saham itu berdasarkan info yang sudah diterimanya?. Sebelum mengambil tindakan, hendaknya dia sudah memastikan bahwa info tersebut adalah otentik. Andaikan ada seorang pesaing yang bisa menyadap dan mengganti isi pesan yang terkirim (misalkan mengganti simbolnya saja), tentunya si penyadap ini bisa memancing dan membohongi pialang tersebut, sehingga membeli saham yang salah. Isu utamanya adalah : Bagaimana memastikan bahwa konten yang sampai ke *mobile device* adalah benar otentik dan tidak berubah dari tempat bursa hingga ke pengguna?

Maka demikian, Integritas data adalah salah satu aspek terpenting dari kemanan komunikasi. Secara fisik, untuk mewujudkan jaringan yang benar-benar aman dibutuhkan investasi mahal dan terbatas pada jangkauan wilayah tertentu saja. Jika bisnis harus berjalan dalam sebuah infrastruktur Internet sebagai media utamanya dalam komunikasi data, bisnis itu harus menghadapi *constraints* bahwa Internet adalah sangat tidak aman, paket data yang ditransmisikan harus melewati banyak *router* yang dilalui, yang mana diluar tanggung jawab pengirim. Sifat dari komunikasi data lewat internet adalah sangat *vulnerable*.

Mekanisme pengamanan yang digunakan dalam *wireless web services* adalah *PKI* (Infrastruktur kunci publik) dan tanda-tangan digital. Secara umum tanda-tangan digital dengan infrastruktur kunci publik yaitu, setiap orang memiliki 2 kunci kriptografi, kunci privat dan kunci publik. Tentunya kunci publik tidak rahasia, kunci privat rahasia. Sebuah pesan yang dienkripsi dengan kunci privat, hanya bisa didekripsi dengan kunci publik yang bersesuaian. Ketika pengirim mengirim pesan, pesan sudah ditambah dengan tanda-tangan digital, yaitu enkripsi dari fungsi *hash* sebuah konten. Penerima akan mendekripsi tanda-tangan tersebut dengan kunci publik yang bersesuaian. Jika sesuai, maka dapat dipastikan bahwa pesannya otentik. Karena pesan aslinya bisa berukuran panjang, dan algoritma untuk membangkitakan dan memverifikasi tergantung dari panjang pesan, maka digunakan versi pendek dari pesan, yang biasa dikenal dengan *digest*, *digest* inilah yang kemudian akan dienkripsi. *Digest* berukuran sama untuk semua pesan, dibentuk dari fungsi *hash* satu arah dari pesan apapun, perhitungan digest ini sangat cepat dan tidak membutuhkan *resource device* yang banyak. Proses verifikasi diawali dengan pemerksaan digest dari pesan yang diterima, jika digest nya tidak cocok, maka pesan sudah

divonis tidak valid sebelum dilakukan dekripsi. Dalam aplikasi sebenarnya, kunci publik sendiri tersertifikasi oleh Lembaga otoritas sendiri untuk memvalidasi identitas dari pengirim. Tapi dalam makalah ini tidak dijelaskan penanganan sertifikasi digital, diasumsikan bahwa setiap pengirim adalah terpercaya dan menggunakan kunci publik yang belum didaftarkan ke lembaga tersebut, agar memudahkan ilustrasi dari aplikasi.

## B. *XML Digital Signature*

Seperti telah dijelaskan sebelumnya, *XML* telah menjadi protokol pertukaran data utama dalam dunial *web services*. Pesan *XML* yang melalui *web services* harus melalui banyak perantara sebelum mencapai tujuannya. Jadi adalah penting untuk mengamankan komten yang dipertukarkan dari titik satu ke titik lain, hingga ke tujuannya. Cara yang terbaik adalah dengan melampirkan dokumen *XML* yang berisi infornasi keamanan (seperti tanda-tangan, *digest*, kunci publik, dan lain-lain) bersama dengan konten yang ditransmisikan.

*XML Digital Signature* adalah spesifikasi *W3C* untuk menambahkan sebuah tanda-tangan digital ke dokumen *XML*. Pengirim bisa memilih untuk menandatangani keseluruhan dokumen atau hanya sebagian saja. Tanda-tangan digital, *digest*, dan kunci publik yang akan dibutuhkan untuk mendekripsi akan diformat menjadi elemen-elemen *XML*. Elemen-elemen tambahan ini bisa membungkus pesan atau disisipkan kedalam pesan asli, dalam makalah ini akan digunakan yang pertama.

**1. Menangani *XML digital signature* dalam aplikasi *MIDP***

IBM alphaWorks telah mengembangkan sebuah *package* Java yang disebut *XML Security Suite* yang mendukung spesifikasi *XML digital signature* yang terbaru. JSR 105 adalah sebuah spesifikasi *API* Java yang dikembangkan unutk membuat sebuah standar untuk memproses tanda-tangan digital. Namun, *API* ini hanya bisa digunakan dalam *J2SE* yang berarti *API* ini bisa digunakan dalam *server side* untuk memproses *XML* digital-signature, tapi tidak untuk wireless device (front-end).Untuk menangani *XML* digital signature, wireless device yang akan digunakan harus mendukung fungsi-fungsi dibawah ini:

1. Membaca dan menulis data dari sebuah dokumen *XML*.

Dari contoh yang akan diberikan, dokumen *XML* yang akan diproses, akan di-*parsing* menjadi objek dalam Java dengan *kXML parser*, yang merupakan *MIDP XML parser* yang didapat dari internet.

2. Membuat dan memverifikasi tanda-tangan

Fungsi ini membutuhkan *API* kriptografi yang tidak ada dalam spesifikasi *MIDP 1.0*. Di bagian selanjutnya akan dijelaskan sebuah *package* kriptografi yang bisa digunakan di kedua sisi untuk membuat dan menandatangani *XML digital signature*

**2. *Digital-signatures* dalam sebuah aplikasi *mobile* yang aman**

Tanda-tangan digital tidak hanya digunakan dalam pengamanan data yang ditransmisikan selama komunikasi, tetapi juga untuk mengamankan aplikasinya itu sendiri. *Wireless-device* sering digunakan untuk mengunduh aplikasi *mobile* lewat jaringan internet sebelum digunakan. Ada kemungkinan pihak tertentu menyadap pendistribusaian aplikasi *mobile* tersebut, dan memasukkan virus dan *trojan* berbahaya misalnya. Bagaimana bisa dipastikan bahwa aplikasi tersebut adalah otentik?. Penyedia aplikasi tersebut bisa menandatangani keseluruhan file .JAR dan .JAD dengan setifikat digital. Pengguna *wireless* bisa menjamin keaslian dan keamanan dari aplikasi.

## C. *API Bouncy-Castle Crypto*

*Bouncy-Castle* adalah *opensource*, sebuah *package* yang sederhana dan menyediakan implementasi untuk *Java Cryptography Extension (JCE 1.2.1)*. Karena *Bouncy-Castle* dirancang cukup sederhana dan cukup ringan, bisa dijalankan di J2SE 1.4 hingga *J2ME* (termasuk plattform *MIDP*). Sejauh ini yang penulis eksplorasi, adalah satu-satunya *API* kriptografi yang berjalan di *MIDP*.

Meski cukup bagus, ada bebrapa masalah utama dengan *Bouncy-Castle* ini, yaitu kurangnya dokumentasi. Dokumentasinya

belum cukup baik meski selalu diperbaharui, JavaDoc nya juga belum ditulis dengan baik. Hampir sama dengan JCE dan *package* kriptografi lainnya, *Bouncy-Castle* menggunakan tipe *polymorphism* untuk memisahkan konsep umum dari implementasi algoritmanya. Agak susah memang untuk memahami hubungan antar-kelas dan mengenali tipe yang tepat untuk parameter dari sebuah *method* dan *return value* nya.

## D. Implementasi *XML Digital Signatures*

Kita telah membicarakan teknologi dan konsep pengamanan dalam *wireless web services* dalam uraian diatas. Penjelasan selanjutnya akan menjelaskan keseluruhan proses dari pengamanan wireless *web services* berikut contoh aplikasinya. Pejelasannya mencakup pembangkitan sepasang kunci, penandatanganan dokumen di sisi *server*, bagaimana melakukan proses *encode* dan mentransmisikan dalam sebuah format dokumen *XML* dan verifikasi dokumen di sisi *client.*

Secara ringkas, kesemua proses dapat diuraikan sebagai berikut:

1. *Server* membangkitkan sepasang kunci yang terdiri dari sebuah kunci publik acak dan privat menggunakan beberapa fungsi yang ada dalam *API*. Dalam contoh aplikasi sebenarnya dalam *web services*, langkah ini biasanya tidak perlu,karena pasangan kunci biasanya sudah dibagkitkan dan disimpan dalam *database* kunci *server*.

2. Ketika *client* mengakses informasi dari sebuah halaman *JSP*, *server* menghitung *digest* dari pesan yang diakses.

3. Halaman *JSP* terebut kemudian membangkitkan sebuah tanda-tangan digital dari digest, menggunakan kunci privat *(signing mode)*

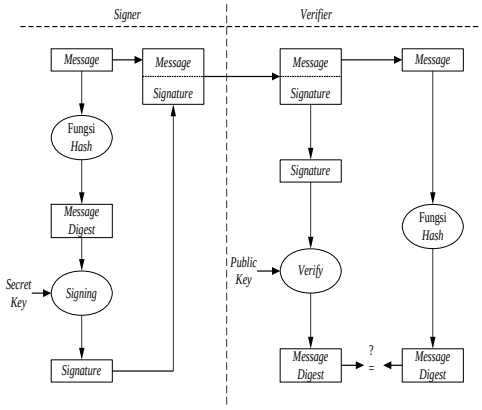
4. *Server* kemudian melekatkan informasi tanda-tangan, termasuk *digest* nya dan kunci publik sebagai parameter dalam dokumen pesan *XML*.

5. Dokumen yang telah ditransmisikan, akan diproses oleh *wireless device*. Dokumen *XML* kemudian akan di-*parsing*, dan dikelompokkan *digest*, kunci publik dan tanda-tangan digital nya.

6. *Client* akan menghitung digest dari pesan asli yang sudah diparsing dari dokumen *XML*, dan kemudian dibandingkan dengan *digest* dari *server*. Jika kedua digest tidak cocok, verifikasi dokumen dinyatakan gagal, dan proses otentikasi berhenti disini, sebaliknya jika digestnya cocok masuk ke langkah selanjutnya.

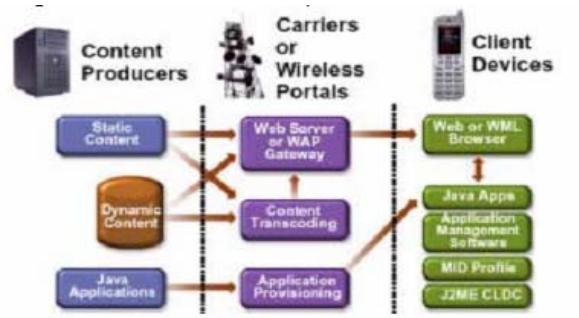
7. *Client* merangkai kembali kunci publik dari sejumlah parameter yang disertakan dalam dokumen *XML* tersebut.

8. *Client* mendekripsi tanda-tangan dengan kunci-publik untuk memverifikasi tanda-tangan.




**Gambar 1 Otentikasi dengan sidik-dijital menggunakan fungsi hash satu-arah**

Di gambar 1 terlihat mekanisme otentikasi tanda tangan, dalam hal ini *signer* adalah *server*, dan *verifier* adalah *client* yang menggunakan *mobile device*. Kita akan mengimplementasikan contoh sederhana dari mekanisme otentikasi dalam *wireless web services* di bagian selanjutnya. Karena contoh yang diberikan menggunakan *API* Bouncy-Castle di kedua *plattform*, baik itu di sisi *server* maupun sisi *client*, maka akan sangat memudahkan dalam penandatanganan dokumen di sisi *server* dan pemverifikasi dokumen di sisi *client*.




**Gambar 2 Proses *retrieving* konten oleh mobile device**

Bagaimana sebuah konten dapat di-*retrieve* oleh *mobile device* ?. Gambar 2 bisa menjelaskan secara singkat proses *retrieval*-nya. Pada bagian atas kiri terlihat *content* berisi *static file* atau bisa juga berupa *dynamic content* atau juga Java *Application* yang di-*generated* oleh *server*. Lebih luas lagi, informasi dapat diekstrak dari basis data dan dimodifikasi menggunakan *servlets*, *EJB*, atau teknologi *server-side* lainnya. Selanjutnya kita lihat pada bagian tengah, Java *Application* ditampilkan oleh *Web Server* dan seterusnya ditransmisikan melalui jaringan *wireless* menuju *mobile devices*. *Web server* atau *WAP gateway system* akan mendistribusikan *content* pada *mobile phone* atau *device* lainnya. Akhirnya *content* diterima oleh client device dengan melakukan proses *transcoding*, baik itu mengambil data dari basis data, atau *XML*, atau *HTML*, dan selanjutnya diterjemahkan kedalam format *WML* atau *cHTML* bergantung pada format yang kita punyai, di aplikasi ini menggunakan *XML* yang sudah ditentukan sebelumnya, elemen-elemen *XML* nya juga proses *parsing* dokumen berdasarkan *API* tertentu.

**1. Membuat *digest* dari pesan.**

Algoritma yang akan digunakan adalah *SHA-1*, implementasi algoritma ini sudah ada dalam *API Bouncy-Castle*, kita hanya akan membuat tanda-tangan yang sesuai dengan format *XML*. Fungsi yang akan dibuat yaitu sebuah fungsi `getDigest()`, masukannya adalah pesan berukuran maksimum $2^{64}$ bit dan keluarannya adalah sebuah *message digest* berukuran 160 bit. Berikut ini adalah kode fungsi pembangkitan *digest*.

```
static public String getDigest( String mesg ) throws Exception {
  //membuat objek digest dari konstruktor
  SHA1Digest digEng = new SHA1Digest();

  //pesan yang telah diubah dari string menjadi array of bytes
  byte [] mesgBytes = mesg.getBytes();

  //menginisialisasi digest dengan ukuran panjang pesan
  digEng.update( mesgBytes, 0, mesgBytes.length );

  //array of bytes untuk menampung hasil byte per byte digest
  byte [] digest = new byte[digEng.getDigestSize()];

  //finalisisasi digest
  digEng.doFinal(digest, 0);

  // mengubah digest dari format ASCII menjadi XML
  return (new String(Base64.encode(digest)));
}
```

Dalam bagian selanjutnya akan dijelaskan bagaimana menandatangani sebuah pesan dengan *API Bouncy-Castle*, ada dua algoritma yang akan dibandingkan yaitu *DSA* dan *RSA*. Kedua algoritma ini menggunakan algoritma yang sama sekali berbeda dan menggunakan parameter yang berbeda. Jadi akan ada beberapa contoh dokumen *XML* untuk pengujian kedua algoritma tersebut, di akhir penjelasan akan dibandingkan keduanya dan saran pengembangan kedepannya.

**2. *DSA Signature***

Secara umum *DSA* adalah suatu algoritma untuk menandatangani pesan. *DSA* tidak dapat digunakan untuk enkripsi; *DSA* dispesifikasikan khusus untuk tanda-tangan digital. *DSA* mempunyai fungsi utama:

1. Pembentukan tanda-tangan (*signature generation*), dan
2. Pemeriksaan keabsahan tanda-tangan (*signatire verification*).

Sebagaimana halnya pada algoritma kriptografi kunci publik, *DSA* menggunakan dua buah kunci, yaitu kunci publik dan kunci privat, sedangkan verifikasi tanda-tangan menggunakan kunci publik pengirim. dalam contoh aplikasi ini, parameter-parameter pembangun kunci publik, ikut disertakan dalam tanda tangan.

***Parameter DSA***

1. *p*, adalah bilangan prima dengan panjang *L* bit, yang dalam hal ini $512 \leq L \leq 1024$, parameter ini sifatnya publik, kondisi inilah yang menyebabkan parameter ini disertakan dalam *XML Digital Signatures*
2. *q*, adalah bilangan prima 160 bit, merupakan faktor dari $p - 1$. parameter ini juga publik, ikut disertakan juga dalam *XML Digital Signatures*
3. $g = h^{(p-1)/q} \bmod p$, yang dalam hal ini $h < p - 1$, sedemikian hingga $h^{(p-1)/q} \bmod p > 1$. parameter ini juga bersifat publik.
4. *x*, adalah bilangan bulat kurang dari *q*. Parameter *x* adalah kunci privat.
5. *m*, pesan yang akan diberi tanda-tangan.

Kunci publik dinyatakan sebagai (p,q,g,y), dan kunci privat dinyatakan sebagai (p,q,g,x);

Untuk membuat sebuah tanda tangan *DSA*, dibutuhkan sepasang kunci. Fasilitas ini disediakan oleh *API* yaitu dengan menggunakan method `DSASigUtil.generateKeys()`untuk membangkitkan sepasang kunci. Seperti sudah dijelaskan sebelumnya, dalam aplikasi *web services* nyata, langkah ini biasanya dilakukan oleh *server* secara *offline*, (kunci disimpan dalam *server database*). Tanda tangan yang sudah dibangkitkan dibungkus bersama dokumen *XML* yang berisi pesan, tanda-tangan dan parameter-parameter kunci lainnya. Kode untuk membuat sebuah tanda-tangan digital ditunjukkan dibawah ini.

```
// Membangkitkan sebuah bilangan acak
SecureRandom sr = new SecureRandom();
// Membangkitkan parameter-parameter DSA.
DSAParametersGenerator DSAParaGen = new DSAParametersGenerator();
DSAParaGen.init(1024, 80, sr);
 DSAPara = DSAParaGen.generateParameters();

// Mendapatkan parameter pembagkitan kunci untuk DSA.
DSAKeyGenerationParameters DSAKeyGenPara = new DSAKeyGenerationParameters(sr, DSAPara);

// Membangkitkan kunci dari parameter sebelumnya.
DSAKeyPairGenerator DSAKeyPairGen = new DSAKeyPairGenerator();
DSAKeyPairGen.init( DSAKeyGenPara );
AsymmetricCipherKeyPair keyPair = DSAKeyPairGen.generateKeyPair();

//menampung dalam kunci publik dan kunci privat yang siap digunakan
privKey = (DSAPrivateKeyParameters) keyPair.getPrivate();
pubKey = (DSAPublicKeyParameters) keyPair.getPublic();
```

Kunci publik yang sudah dibangkitkan ditandai dengan parameter Y, parameter ini bisa didapat dengan memanggil method `pubKey.getY()`. Parameter lainnya G, P dan Q bisa didapat dengan memanggil method dalam kelas `DSAPara` tadi, kode berikut menunjukkan cara untuk mendapatkan parameter lain dalam *DSA*, parameter-parameter ini akan digunakan untuk membuat kembali kunci publik untuk mendekripsi di sisi klien.

```
//semua parameter bertipe string dan sudah di encode dalam format XML
public static String getG() throws Exception {
  return (new String(Base64.encode(DSAPara.getG().toByteArray())));
}
public static String getP() throws Exception {
  return (new String(Base64.encode(DSAPara.getP().toByteArray())));
}
public static String getQ() throws Exception {
  return (new String(Base64.encode(DSAPara.getQ().toByteArray())));
}
public static String getY() throws Exception {
  return (new String(Base64.encode(pubKey.getY().toByteArray())));
}
```

Dengan menggunakan kunci privat, dapat membuat tanda-tangan *DSA* yang terdiri dari 2 buah, R dan S.

```
static public String [] getSignature (String digest) throws Exception {
  // Menandatangani digest
  DSASigner signer = new DSASigner();
  // kunci privat menjadi parameter penyandian
  signer.init( true, privKey );
  // menampung hasil signature dalam array of BigInteger (R dan S)
  BigInteger [] sigArray = signer.generateSignature( digest.getBytes());
  //membuat string penampung untuk R dan S
  String [] result = new String [2];
  // menampung R dalam array result
  result[0] = new String(Base64.encode(sigArray[0].toByteArray()));
  // menampung S dalam array result
  result[1] = new String(Base64.encode(sigArray[1].toByteArray()));

  return result;
}
```

*Server* akan mengenkripsi *digest*, tanda-tangan dan parameter kunci kedalam teks berformat *ASCII* dan melekatkan dalam format *XML digital signature*, contoh hasil dokumen *XML*-nya sebagai berikut :

```
<SignedMesg>
  <mesg>Hello World</mesg>
  <Signature>
    <SignedInfo>
      <SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
      <DigestValue>Ck1VqNd45QIvq3AZd8XYQLvEhtA=</DigestValue>
    </SignedInfo>
    <SignatureValue>
      <R>AMfVKyIUyPGdeUCtJxU+N9kQJc2x</R>
      <S>RwGahqpopPx//bMYXzH8dtY0lhA=</S>
    </SignatureValue>
    <KeyInfo>
      <KeyValue>
        <DSAKeyValue>
          <G>
            FgLTXVdxKAmDQtQHkDdFF5zthKSpQhUCzRgXxz7yzxM
            OLYrRoj5D8AXdGLS+5CzT4gu55MbO62dBfyEWKbWTIO
            6E+CuOfa53wvqjMl67tGxc8szgWWA6ZvRwVVVmJ6wqB
            m5hNLr7q1X2eJKQ+u3XYpFflJktOjV8O3zeEPOtsTQ=
          </G>
          <P>
            AOAu2WqVEKGTF8Zcxgde4vxc8f/Z+hk8A10M0AtY2lU
            8CX54dz2MuD6hOmhqGXJxIVlV9085d9D0yHcMv2wl9V
            Vt0/ww+aqFukCKZj9fHgZzq26nOBXMqibDo67J2vfQw
            EZMvCnyBXdS665whjzl5i7ubXu2Su+AqsodnvG9pyYB
          </P>
          <Q>AMjJUZy1RnQRqe/22BS83k2Hk8VR</Q>
          <Y>
            AM/9leouAW7nyON24xeqibMUpVOW8RyzcdNjp9NiPdfm
            HT42BvB4JL/cXx0tCbyHtcR5G+vALoOo7Mh3JJ+/gjx7
```

```
            sS8uHNngqx6O6dADrc9VdPvyllNDR0szLja1RTRCIy9M
            8p0dKe/U8iotAj2zctjfbrroMu/fTOBhkvb2gVvR
          </Y>
          </DSAKeyValue>
        </KeyValue>
      </KeyInfo>
    </Signature>
</SignedMesg>
```

Proses selanjutnya adalah verifikasi dokumen yang berlangsung dalam aplikasi *MIDP*, *mobile devices* akan mem-*parsing digest*, membuat kembali kunci-publik menggunakan parameter yang sudah disertakan dalam *XML*, dan menggunakan kunci publik tersebut untuk memvalidasi dokumen. Potongan kode verifikasi di *MIDP* ditunjukkan dibawah ini.

```
//fungsi verify menerima input digest, signature dan parameter-//parameter kunci lainnya
static public boolean verify (String digest,
                              String sig_r, String sig_s,
                              String key_g, String key_p,
                              String key_q, String key_y ) {

  BigInteger g = new BigInteger( Base64.decode(key_g) );
  BigInteger p = new BigInteger( Base64.decode(key_p) );
  BigInteger q = new BigInteger( Base64.decode(key_q) );
  BigInteger y = new BigInteger( Base64.decode(key_y) );
  BigInteger r = new BigInteger( Base64.decode(sig_r) );
  BigInteger s = new BigInteger( Base64.decode(sig_s) );

  DSAParameters DSAPara = new DSAParameters(p, q, g);
  DSAPublicKeyParameters DSAPubKeyPara = new DSAPublicKeyParameters(y,
                                          DSAPara);

// Verifikasi
  DSASigner signer = new DSASigner();
//inisialisasi dengan false
  signer.init( false, DSAPubKeyPara );
//pencocokan dokumen dari digest yang sudah dihitung dan kedua tanda  //tangan (R dan S)
  boolean result = signer.verifySignature( digest.getBytes(), r, s );
  return result;
}
```

**3. RSA *Signature***

*RSA* adalah salah-satu algoritma kriptografi kunci publik yang banyak digunakan. Keamanan algoritma *RSA* terletak pada sulitnya memfaktorkan bilangan besar menjadi faktor-faktor prima. Pemfaktoran dilakukan untuk mendapatkan kunci privat. Selama pemfaktoran bilangan besar menjadi faktor-faktor prima belum ditemukan algoritma yang mangkus, maka selama itu kemanan algoritma *RSA* tetap terjamin.

**Parameter *RSA***

1. *p* dan *q* adalah bilangan prima, parameter ini sifatnya rahasia.
2. *n* = p x q, parameter ini sifatnya tidak rahasia
3. $\Phi$ (n) = (*p* -1)(*q*-1), parameter ini sifatnya rahasia
4. *e* (kunci enkripsi), parameter ini sifatnya tidak rahasia
5. *d* (kunci dekripsi), parameter ini sifatnya rahasia

Implementasi dari parameter-parameter *RSA* tersebut ada dalam *API* yang digunakan. Kunci publik dinyatakan sebagai (d,n), dan kunci privat dinyatakan sebagai (e,n). Dalam Implementasinya, algoritma *RSA* hanya mempunyai satu model parameter yaitu *Exponent.* dalam hal ini *Exponent* adalah parameter RSA yaitu *e*, sedangkan *modulus* adalah nilai dari *n*. Contoh pembangkitan *public Exponent* (*pubExp*) ditunjukkan dalam kode berikut.

```
//membangkitkan BigInteger eksponen publik (d)
private static BigInteger pubExp = new BigInteger("11", 16);
```

Pembangkitan sepasang kunci menggunakan *method* `RSASigUtil.generateKeys()` yang ada dalam kelas RSA, *method* ini menggunakan *public Exponent*, yang sudah dibangkitkan lagi, sekali lagi ditekankan bahwa dalam aplikasi *web services* nyata, langkah ini biasanya dilakukan oleh *server* secara *offline*, (kunci disimpan dalam *server database*).

```
//membangkitkan bilangan acak
SecureRandom sr = new SecureRandom();

//membuat parameter kunci dari RSA
RSAKeyGenerationParameters RSAKeyGenPara =
       new RSAKeyGenerationParameters(pubExp, sr, 1024, 80);

//membuat sepasang kunci dari parameter kunci
RSAKeyPairGenerator RSAKeyPairGen = new RSAKeyPairGenerator();
RSAKeyPairGen.init(RSAKeyGenPara);
AsymmetricCipherKeyPair keyPair = RSAKeyPairGen.generateKeyPair();

//membuat kunci privat dan kunci publik
privKey = (RSAPrivateCrtKeyParameters) keyPair.getPrivate();
pubKey = (RSAKeyParameters) keyPair.getPublic();
```

Kunci publik ditandai dengan sebuah parameter bernama `Modulus` dalam hal ini modulus adalah nilai dari *e*, parameter ini bisa didapatkan dengan menggunakan `pubKey.getModulus()`, potongan kode berikut menunjukkan metode yang ada dalam kelas `RSAUtil` yang akan digunakan untuk mendapatkan `Exponent` dan `Modulus`, parameter inilah yang akan digunakan untuk membangun kembali kunci publik di sisi klien.

```
// mengambil nilai modulus (n), dan mengembalikan kedalam tipe string
public static String getMod() throws Exception {
  return (new String(Base64.encode(pubKey.getModulus().toByteArray())));
}

// mengambil parameter umum lainnya (pubExp)
public static String getPubExp() throws Exception {
  return (new String(Base64.encode(pubExp.toByteArray())));
}
```

Dengan menggunakan kunci privat yang telah dibangkitkan sebelumnya, akan digunakan *method* yang ada dalam kelas `RSASigUtil` untuk mendapatkan *signature*.

```
static public String getSignature (String mesg) throws Exception {
  SHA1Digest digEng = new SHA1Digest();
  RSAEngine rsaEng = new RSAEngine();

  PSSSigner signer = new PSSSigner(rsaEng, digEng, 64);
  signer.init(true, privKey);

  byte [] sig = signer.generateSignature( mesg.getBytes() );
  String result = new String( Base64.encode(sig) );
  return result;}
```

*Server* mengenkripsi *digest*, tanda-tangan dan parameter kunci kedalam teks berformat ASCII dan melekatkan dalam format *XML digital signature*, contoh hasil dokumen *XML* nya sebagai berikut :

```
<SignedMesg>
  <mesg>Hello Feb!</mesg>
  <Signature>
    <SignedInfo>
      <SignatureMethod Algorithm="rsa-sha1" />
      <DigestValue>Ck1VqNd45QIvq3AZd8XYQLvEhtA=</DigestValue>
    </SignedInfo>
    <SignatureValue>
      IhJ/UMitJX7sWbzhnG8UKIdDYiZ0mfOUoAwemGiG08C
      WcQ3cUszgJXoIclHW/LN7w54w2FQyLStB+hPKASEC6r
      OjjgTBs6pwhjHCh2XxWx7hS7fdi9/Qk/ybH6xYGaeaZ
      3oHDBjFz3hEDtrvBYcHn3keCavncE22idRX7kBl8Do=
    </SignatureValue>
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus>
            AKT1SyxSm4uT1zYWEPY9IaFY7vDhpkIM7FZeIQ
            OGnKeSEE5d3sPfONkCiHfO2oe4x6jNCXg/ngRi
            tmixBkjfKgHzF4trZZtNQZjfzAgcXGljzp9MD2
            ZEWQbHKvMZvZyJVrT2SlxLzusxWLwXdacprIDG
            bqDAmldBOBpkmrUdPpF9
          </Modulus>
          <Exponent>EQ==</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</SignedMesg>
```

Proses selanjutnya adalah verifikasi dokumen yang berlangsung dalam aplikasi *MIDP*, *mobile devices* akan mem-*parsing*, digest, membuat kembali kunci-publik menggunakan parameter yang sudah disertakan dalam *XML*, dan menggunakan kunci publik tersebut untuk memvalidasi dokumen. Potongan kode verifikasi di *MIDP* ditunjukkan dibawah ini.

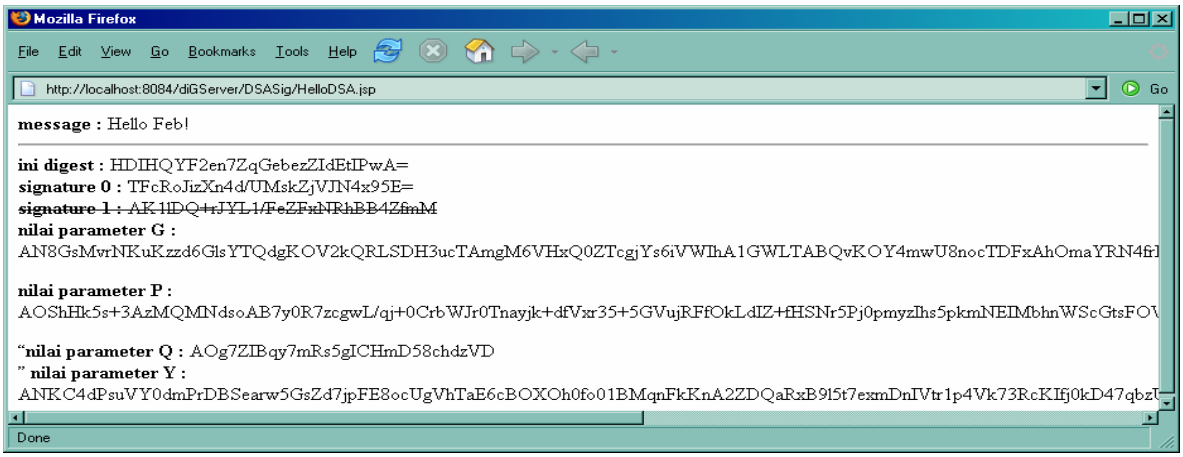
```
//melakukan verifikasi terhadap pesan, prameter-parameternya terdiri //dari pesan, tanda-
tangan, nilai modulus (n), dan eksponen (e)
static public boolean verify (String mesg, String signature,
                              String mod, String pubExp) {
//nilai n
  BigInteger modulus = new BigInteger( Base64.decode(mod) );

//nilai d
  BigInteger exponent = new BigInteger( Base64.decode(pubExp) );

//membangkitkan digest dari pesan
  SHA1Digest digEng = new SHA1Digest();

//inisialisasi parameter RSA
  RSAEngine rsaEng = new RSAEngine();

//membentuk kunci publik dari parameter
  RSAKeyParameters pubKey = new RSAKeyParameters(false, modulus, exponent);

//inisialisasi dengan false
  PSSSigner signer = new PSSSigner(rsaEng, digEng, 64);
  signer.init(false, pubKey);

//hasil verifikasi, membandingkan hasil digest yang sudah dienkripsi //dengan signature
  boolean res = signer.verifySignature( mesg.getBytes(),
                                        Base64.decode(signature) );
  return res;
}
```

**4. Kode program di sisi *server* untuk DSA**

```
<!-- file helloDSA.jsp -->
```

```
<%@ page import="XMLDsig.*" %>
<%
  // Inisialisasi kunci.
  DSASigUtil.generateKeys();
  // Pesan yang akan ditransmisikan
  String mesg = "Hello Feb!";
  // mendapatkan digest dari pesan
  String digest = DSASigUtil.getDigest(mesg);
  // Mendapatkan tanda-tangan
  String [] signature = DSASigUtil.getSignature(digest);
%>
<SignedMesg>
  <mesg><%=mesg%><HR></mesg>
  <Signature>
    <SignedInfo>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
      <DigestValue><%=digest%></DigestValue>
    </SignedInfo>
    <SignatureValue>
      <R><%=signature[0]%></R>
      <S><%=signature[1]%></S>
    </SignatureValue>
    <KeyInfo>
      <KeyValue>
        <DSAKeyValue>
          <G><%=DSASigUtil.getG()%></G>
          <P><%=DSASigUtil.getP()%></P>
          <Q><%=DSASigUtil.getQ()%></Q>
          <Y><%=DSASigUtil.getY()%></Y>
        </DSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</SignedMesg>
```

TampilanHelloDSA.jsp

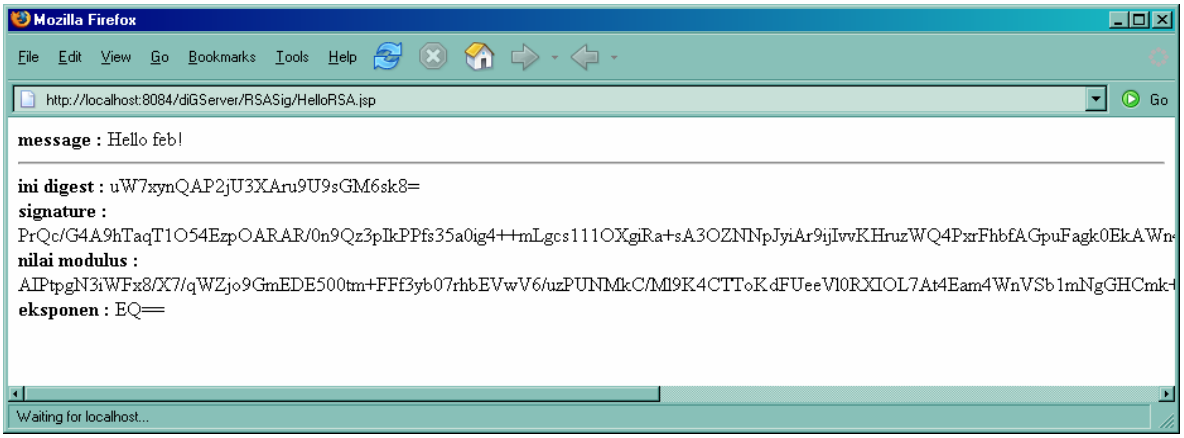



## 5. Kode program di sisi *server* untuk RSA

```
<!-- file helloRSA.jsp -->
<%@ page import="XMLDsig.*" %>
<%
  // Inisialisasi kunci.
  RSASigUtil.generateKeys();

  // Pesan yang akan ditransmisikan
  String mesg = "Hello feb!";

  // mendapatkan digest dari pesan
```

```
  String digest = RSASigUtil.getDigest(mesg);

  // mendapatkan tanda-tangan.
  String signature = RSASigUtil.getSignature(mesg);
%>
<SignedMesg>
  <mesg><%=mesg%></mesg>
  <Signature>
    <SignedInfo>
      <SignatureMethod Algorithm="rsa-sha1" />
      <DigestValue><%=digest%></DigestValue>
    </SignedInfo>
    <SignatureValue><%=signature%></SignatureValue>
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus><%=RSASigUtil.getMod()%></Modulus>
          <Exponent><%=RSASigUtil.getPubExp()%></Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</SignedMesg>
```

TampilanHelloRSA.jsp

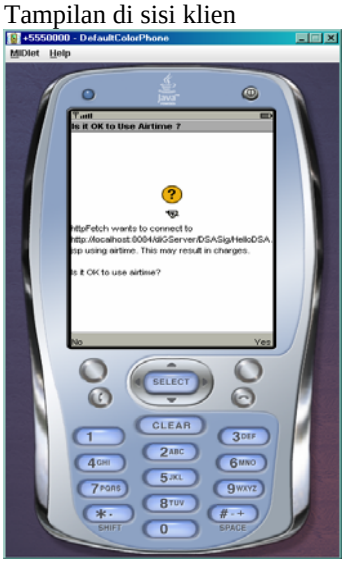



## 6. Kode program di sisi klien untuk DSA

```
public class DSASigTest extends MIDlet implements CommandListener {

....
  /**
   *event handler utama dalam aplikasi
   **/
  public void commandAction(Command command, Displayable screen) {

    if (command == cancelCommand) {
      // keluar dari midlet,jika menekan tombol quit
      destroyApp(false);
      notifyDestroyed();
    } else if (command == fetchCommand) {//melakukan fetch ke server
      try {
        // Mendapatkan URL dari textfield
        //String url = textField.getString();
        url = url.trim();

        HttpConnection conn = (HttpConnection) Connector.open(url,Connector.READ);
        conn.setRequestMethod(HttpConnection.GET);
```

```
        // mendapatkan response stream dari server
        DataInputStream is = conn.openDataInputStream();
        InputStreamReader reader = new InputStreamReader(is);

        //membuat objek XML dari dokumen yang di fetch
        XmlParser parser = new XmlParser (reader);
        Document doc = new Document();
        doc.parse(parser);
        //parsing XML, mengelompokkan berdasar tag
        Element signedMesg = doc.getRootElement();

        String mesg = signedMesg.getElement("mesg").getText();
        String digest =
signedMesg.getElement("Signature").getElement("SignedInfo").getElement("DigestValue").get
Text();

        String key_g =
signedMesg.getElement("Signature").getElement("KeyInfo").getElement("KeyValue").getElemen
t("DSAKeyValue").getElement("G").getText();
        String key_p =
signedMesg.getElement("Signature").getElement("KeyInfo").getElement("KeyValue").getElemen
t("DSAKeyValue").getElement("P").getText();
        String key_q =
signedMesg.getElement("Signature").getElement("KeyInfo").getElement("KeyValue").getElemen
t("DSAKeyValue").getElement("Q").getText();
        String key_y =
signedMesg.getElement("Signature").getElement("KeyInfo").getElement("KeyValue").getElemen
t("DSAKeyValue").getElement("Y").getText();

        String sig_r =
signedMesg.getElement("Signature").getElement("SignatureValue").getElement("R").getText()
;
        String sig_s =
signedMesg.getElement("Signature").getElement("SignatureValue").getElement("S").getText()
;

        // tutup koneksi
        is.close();
        //verifikasi
        if ( digest.equals(DSASigUtil.getDigest(mesg)) ) {
          mesg = mesg + "\nDigest valid";
          if ( DSASigUtil.verify(digest, sig_r, sig_s,
                                 key_g, key_p, key_q, key_y) ) {
            mesg = mesg + "\nSignature valid";
          } else {
            mesg = mesg + "\nSignature salah, pesan berubah, atau kunci tidak sesuai";
          }
        } else {
          mesg = mesg + "\nDigest is failed";
        }
...
}
```

Tampilan di sisi klien




**Gambar 3 ketika *client* ingin mengunduh konten**


**Gambar 4 ketika proses verifikasi berhasil**

### 7. Kode program di sisi klien untuk RSA

```
public class RSASigTest extends MIDlet implements CommandListener {

...
  /**
   *event handler utama dalam aplikasi
   **/
  public void commandAction(Command command, Displayable screen) {

   ...
       ... /*bagian ini sama dengan kode untuk DSA*/
        // mendapatkan response stream dari server
       InputStreamReader reader = new InputStreamReader(is);
       //membuat objek XML dari dokumen yang di fetch
       XmlParser parser = new XmlParser (reader);
       Document doc = new Document();
       doc.parse(parser);
       //parsing XML, mengelompokkan berdasar tag
       Element signedMesg = doc.getRootElement();
        String mesg = signedMesg.getElement("mesg").getText();
```

```
        String                                      digest                                          =
signedMesg.getElement("Signature").getElement("SignedInfo").getElement("DigestValue").get
Text();
        String                                       mod                                            =
signedMesg.getElement("Signature").getElement("KeyInfo").getElement("KeyValue").getElemen
t("RSAKeyValue").getElement("Modulus").getText();
        String                                      pubExp                                          =
signedMesg.getElement("Signature").getElement("KeyInfo").getElement("KeyValue").getElemen
t("RSAKeyValue").getElement("Exponent").getText();
        String                                     signature                                        =
signedMesg.getElement("Signature").getElement("SignatureValue").getText();
        // tutup koneksi
        is.close();
       //verifikasi
        String displayMesg = mesg;
        if ( digest.equals(RSASigUtil.getDigest(mesg)) ) {
          displayMesg = displayMesg + "\nDigest valid";
          if ( RSASigUtil.verify(mesg, signature, mod, pubExp) ) {
            displayMesg = displayMesg + "\nSignature valid";
          } else {
            displayMesg = displayMesg + "\nSignature valid";
          }
        } else {
          displayMesg = displayMesg + "\nSignature salah, pesan berubah, atau kunci tidak
sesuai";}
...
```

Tampilan di sisi klien

**Gambar 5 ketika *client* ingin mengunduh konten**

**Gambar 6 ketika proses verifikasi berhasil**

# E. Hasil pengujian dan pelaporan kinerja

Tes yang dilakukan dengan *J2ME WTK 2.5* emulator menunjukkan bahwa, pembuatan *digest* dan *parsing* dari *XML digital signatures* berlangsung cukup cepat. Kendala utamanya terletak pada algoritma kunci publik yang melibatkan operasi `BigInteger()` pada klien. dan parameter-parameter dari *RSA* dan *DSA* sangat bergantung dari digest yang berukuran tetap. Maka secara langsung dapat dibandingkan kinerja keduanya. Parameter pengujiannya adalah waktu sejak melakukan koneksi pertama kali, hingga mendapatkan *alert* pertama apakah konten yang di-*fetch* dari *mobile devices* valid atau tidak.

*API Bouncy-Castle* menyediakan banyak kelas dari algoritma kunci publik, namun yang digunakan dalam aplikasi ini hanya *DSA* dan *RSA* saja. Namun, tidak semua algoritma kunci publik secara nyaman bisa digunakan dalam aplikasi sebenarnya. Karena *API* ini murni berbasiskan bahasa java, maka sangat tergantung pada JVM yang ada dalam *mobile devices* tersebut. Operasi yang sangat membutuhkan *resources* adalah operasi penghitungan dari `BigInteger()` tanpa ada perlakuan khusus, perlakuan khusus misalnya, pada saat melakukan operasi `BigInteger()` yang krusial, perpangkatan misalnya, ada mekanisme tersendiri dalam pemrograman sistem aplikasi di *mobile*, contohnya *thread programming*, ini juga merupakan saran pengembangan kedepannya.

Hasil pengujian menunjukkan algoritma *RSA* menunjukkan kinerja yang lebih baik dan lebih bisa diterima ketimbang algoritma *DSA*. Pengujian yang dilakukan dengan kunci publik berukuran 1024-bit hanya membutuhkan kurang dari 90 detik sampai proses verifikasinya menunjukkan hasil. Tentunya kinerja ini bisa ditingkatkan dengan memperkecil ukuran kunci kurang dari 1024-bit. Tapi meskipun demikian, seperti saran penulis sebelumnya, perlu ada perlakuan khusus dalam operasi verifikasi ini, misalnya dengan *thread*, kemudian berjalan dalam *background process* dan diberi *progress bar* agar pengguna bisa mengetahui sudah sejauh mana proses verifikasinya.

Sedangkan kinerja dari algoritma *DSA* tidak menunjukkan hasil yang memuaskan dalam implementasinya. Sebauh tanda-tangan dengan ukuran kunci 1024-bit membutuhkan waktu lebih dari 20 menit sampai proses verifikasi nya berhasil, hal ini karena teknik verifikasi dalam algoritma yang melakukan operasi perbandingan yang lebih banyak dari algoritma sebelumnya.

Masalah tentang kinerja secara umum telah mensyaratkan adanya suatu *JVM* yang lebih tangguh, sehingga lebih optimal dalam menangani operasi `BigInteger()`dalam algoritma kunci publik. *JVM* ini harus bisa menjalankan operasi perhitungan bilangan bulat yang rumit terkait dengan kriptografi. *JVM* yang baik juga seharusnya dapat memaksimalkan *hardware* khusus dan fitur yang sudah tersedia dalam *Operating System devices* tersebut untuk meningkatkan kinerja operasi komputasinya. Algoritma kunci publik juga digunakan dalam *handshaking* dalam protokol komunikasi aman lainnya seperti *HTTPS*. Banyak *MIDP* yang sudah mendukung protokol *HTTPS* ini dengan kinerja yang cukup baik. *VM* dari *MIDP4Palm* misalnya, bisa memanfaatkan protokol `inethttps` yang sudah disediakan oleh *Operating System* untuk membuat sebuah jalur komunikasi yang aman. Dan pada beberapa waktu kedepan, dapat dipastikan bahwa teknologi *VM* dalam *mobile devices* dan *library* dari bahasa pemrograman tidak hanya meningkatkan operasi algoritma kunci publik yang terkait dengan keamanan sebuah komunikasi, tetapi juga sekaligus fungsi keamanan secara umum seperti tanda-tangan digital.

# F. Kesimpulan

Kesimpulan yang dapat diambil dari implementasi *XML digital signatures* pada *wireless web services* adalah sebagai berikut :

1. Plattform Java memegang peranan penting dalam pembangunan aplikasi *wireless Web services*. Di sisi klien (*front-end*) digunakan teknologi *J2ME* yang menawarkan kompatibilitas terhadap hampir semua *mobile-devices*, dan *library* yang cukup banyak yang dapat digunakan untuk mngembangkan sebuah aplikasi *wireless Web services* di sisi klien yang cukup kompleks.

2. *Wireless web services* adalah mudah diserang oleh berbagai macam jenis penyerangan. Dan tidak semua protokol *security* cocok diterapkan dalam aplikasi *Wireless web service.* Fokus utamanya adalah pengamanan dari konten itu sendiri, bukan kepada jalur komunikasi yang dilalui untuk mencapai tujuan. Maka *XML Digital signatures* dapat digunakan sebagai mekanisme pengamanan pengiriman data dalam aplikasi *wireless Web servies.*

3. Mekanisme pengamanan yang digunakan dalam *wireless Web services* adalah *PKI* (Infrastruktur kunci publik) dan tanda-tangan digital. Secara umum tanda-tangan digital dengan infrastruktur kunci publik yaitu, setiap orang memiliki 2 kunci kriptografi, kunci privat dan kunci publik.

4. *API Bouncy-Castle* yang digunakan dalam implementasi sangat memudahkan penggunaan Infrastruktur kunci publik dalam pengamanan *wireless Web services*, karena *API* ini bisa diimplementasikan dalam lingkungan pengembangan *J2EE* dan *J2ME* sekaligus.

5. Hasil pengujian menunjukkan RSA lebih bisa diimplementasikan di aplikasi *wireless Web services*, karena kecepatan komputasi yang cukup baik.

6. Kecepatan melakukan proses verifikasi tanda tangan bergantung pada kemampuan mobile device untuk melakukan proses komputasi. Dalam hal ini bergantung pada *JVM* yang ada dalam *mobile devices* tersebut.

# G. DAFTAR PUSTAKA

[1] Yuan, Michael (2002) *general security issues of the J2ME/MIDP platform* .DeveloperWorks.

[2] Tutorial *"The MIDlets advantage"* (2002). DeveloperWorks

[3] Munir, Rinaldi. (2005). Diktat kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[4] The Bouncy Castle Crypto *documentation, http://www.bouncycastle.org/,* Tanggal akses : 4 Desember 2006 pukul 17:00.

[5] *J2ME Web services specification (JSR172), http://jcp.org/jsr/detail/172.jsp,* Tanggal akses: 4 Desember 2006 pukul 17:00.

[6] Loeb, Larry (2001), *XML digital signatures comprehensive article.*

[7] *J2ME XML digital signature specification (JSR105), http://jcp.org/jsr/detail/105.jsp,* Tanggal akses : 4 Desember 2006 pukul 17:00.